BAB II

LANDASAN TEORI

2.1 Konsep Dasar Rekayasa Perangkat Lunak *(Software Engineering)*

Menurut Rosa, dkk (4:2011) "Rekayasa perangkat lunak (*software engineering*) merupakan pembangunan dengan menggunakan prinsip atau konsep rekayasa dengan tujuan menghasilkan perangkat lunak yang bernilai ekonomi yang bisa dipercaya dan bekerja secara efisien menggunakan mesin". Sekilas tentang rekayasa perangkat lunak akan dijelaskan sebagai berikut :

2.1.1 Pengertian Perangkat Lunak

Perangkat lunak (*software*) adalah program komputer yang terasosiasi dengan dokumentasi perangkat lunak seperti dokumentasi kebutuhan, model desain, dan cara penggunaan (*user manual*). Sebuah program komputer tanpa terisosiasi dengan dokumentasinya maka belum dapat disebut perangkat lunak (*software*). Sebuah perangkat lunak juga sering disebut dengan sistem perangkat lunak. Sistem berarti kumpulan komponen yang saling terkait dan mempunyai satu tujuan yang ingin dicapai.

Sistem perangkat lunak berarti sebuah sistem yang memiliki komponen berupa perangkat lunak yang memiliki hubungan satu sama lain untuk memenuhi kebuthan pelanggan (*customer*). Pelanggan (*customer*) adalah orang atau organisasi yang dengan sukarela mengeluarkan uang untuk memesan atau membeli perangkat lunak. *User* atau pemakai perangkat lunak adalah orang yang memiliki kepentingan untuk memakai atau menggunakan perangkat lunak untuk memudahkan pekerjaannya.

2.1.2 Pengertian Rekayasa Perangkat Lunak

Istilah Rekayasa Perangkat Lunak *(RPL)* secara umum disepakati sebagai terjemahan dari *Software Engineering*. Istilah *Software Engineering* mulai digunakan pertama kali pada tahun 1950-an dan awal tagun 1960-an. Pada tahun 1968 dan 1969, komite sains *NATO* mensponsori dua konfrensi tentang rekasaya perangkat lunak yang memberikan dampak kuat terhadap perkembangan rekayasa

perangkat lunak. Sebagian orang mengartikan *RPL* hanya sebatas pada bagaimana membuat program komputer. Padahal ada perbedaan yang mendasar antara perangkat lunak *(software)* dan program komputer. Perangkat lunak *(software)* adalah program komputer yang terasosiasi dengan dokumentasi perangkat lunak seperti dokumentasi kebutuhan, model desain, dan cara penggunaan (*user* manual) (Rosa, dkk, 2011).

Pada definisi Rosa, dkk (4:2011) mengungkapkan "*RPL* (*Software Engineering*) Merupakan pembangunan dengan mengunakan prinsip atau konsep rekayasa dengan tujuan menghasilkan perangkat lunak yang bernilai ekonomi yang dipercaya dan bekerja secara efisien menggunakan mesin".

Rosa, dkk (5:2011) juga mengungkapkan, "perangkat lunak yang baik adalah perangkat lunak yang dapat memenuhi kebutuhan pelanggan *(customer)* atau *user* (pemakai perangkat lunak) atau berorientasi pada pelanggan atau pemakai perangkat lunak, bukan berorientasi pada pembuat atau pengembang perangkat lunak".

2.1.3 Model Rekayasa Perangkat Lunak

Proyek pengembangan perangkat merupakan pekerjaan yang sangat memakan biaya dan waktu karena pengembangan perangkat lunak ini difokuskan pada perencanaan dan pengendalian.

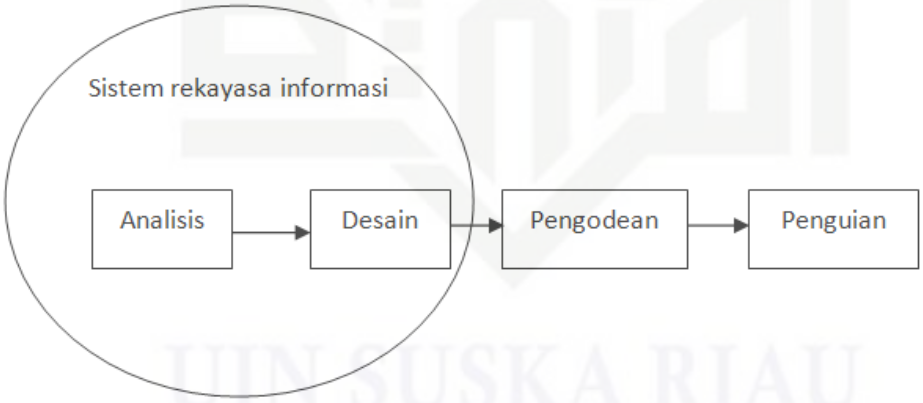
Untuk menyelesaikan masalah aktual di dalam sebuah seting industri, rekayasa perangkat lunak atau tim perekayasa harus menggabungkan strategi pengembangan yang melingkupi lapisan proses, metode, dan alat bantu serta fase-fase generik. Model proses untuk rekayasa perangkat lunak dipilih berdasarkan sifat *aplikasi* dan proyeknya, metode dan alat-alat bantu yang akan dipakai, dan *kontrol* serta penyampaian yang dibutuhkan.

Bagian ini akan membahas secara umum model proses yang sering digunakan dalam komunitas pengembang perangkat lunak. Pembahasan ini akan menggunakan model air terjun (*waterfal model*) dan dilanjutkan dengan pendekatan *prototipe*. Proses serupa dilakukan untuk peningkatan proses pengembangan.

a. Model Air Terjun (*Waterfall Model*)

Nama *model* ini sebenarnya adalah *Linear Sequential Model*. Model ini sering disebut dengan *classic life cycle* atau model *waterfall*. Model ini muncul pertama kali yaitu sekitar tahun 1970 sehingga sering dianggap kuno, tetapi merupakan model yang paling banyak dipakai didalam *Software Engineering (SE)*. Model ini melakukan pendekatan secara sistematis dan urut mulai dari level kebutuhan sistem lalu menuju ke tahap *analisis, desain, coding, testing atau verification*, dan *maintenance*. Disebut dengan *waterfall* karena tahap demi tahap yang dilalui harus menunggu selesainya tahap sebelumnya dan berjalan berurutan. Misal desain harus menunggu selesainya tahap sebelumnya yaitu tahap *requirement*.

"Waterfall Model sering juga disebut model *sekuensial linier (Sequential linear)* atau alur hidup klasik (*Classic life cycle).* Model ini menyediakan pendekatan alur hidup perangkat lunak secara *sekuensial* atau terurut dimulai dari analisis, desain, pengkodean, pengujian dan tahapan pendukung *(support)"* (Rosa, dkk, 26:2011).



Gambar 2.1 *Waterfall Model*
(Sumber: Rosa, dkk, 27:2011)

Gambar 2.1 di atas adalah tahapan umum dari model proses ini. memecah model ini menjadi 4 tahapan meskipun secara garis besar sama dengan tahapan-tahapan model *waterfall* pada umumnya. Berikut adalah penjelasan dari tahap-tahap yang dilakukan di dalam model tersebut:

1. **Analisis Kebutuhan perangkat lunak**

   Proses pengumpulan kebutuhan dilakukan secar intensif untuk menspesifikasikan kebutuhan perangkat lunak agar dapat dipahami perangkat lunak seperti apa yang dibutuhkan oleh *user*. Spesifikasi perangkat lunak pada tahap ini perlu untuk didokumentasikan.

2. **Desain**

   Desain perangkat lunak adalah proses multilangkah yang focus desain pembuatan program perangkat lunak termasuk struktur dan arsitektur perangkat lunak, representasi antar muka, dan proses pengkodean.

3. **Pembuatan kode program**

   Desain harus ditranslasikan ke dalam program perangkat lunak. Hasil dari tahapan ini adalah program computer sesuai dengan desain yang telah dibuat pada tahap desain.

4. **Pengujian**

   Pengujian focus kepada perangkat lunak secara dari segu lojik dan fungsional dan memastikan bahwa semua bagian sudah diuji. Hal ini dilakukan untuk meminimalisir (*error*) dan memastikan keluaran yang dihasilkan sesuai dengan yang diinginkan.

5. **Pendukung (*support)* atau pemeliharaan(*maintenance)***

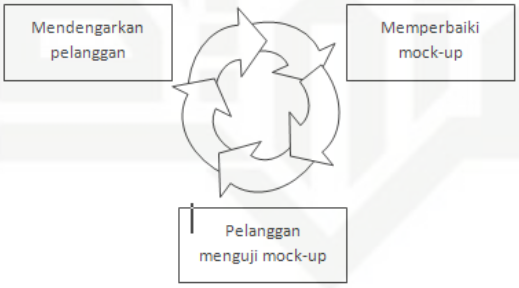
   Tidak menutup kemungkinan sebuah perangkat lunak mengalami perubahan ketika sudah dikirimkan ke user. Perubahan bias terjadi karena adanya kesalahan yang muncul

dan tidak terdeteksi saat pengujian atau perangkat lunak beradap tasi terhadap linkungan baru.

b. Model *Prototipe*

Prototyping Paradigma dimulai dengan pengumpulan kebutuhan. Pengembang dan pelanggan bertemu dan mendefinisikan obyektif keseluruhan dari perangkat lunak, mengidentifikasi segala kebutuhan yang diketahui, dan area garis besar dimana definisi lebih jauh merupakan keharusan kemudian dilakukan perancangan. “Model ini dapat digunakan untuk menyambungkan ketidakpahaman pelanggan mengenai hal teknis dan menjelaskan spesifijasi kebutuhan yang diinginkan pelanggan kepada pengembang perangkat lunak” (Rosa, dkk, 29:2011).

Konsumen potensial menggunakan *prototipe* dan menyediakan masukan untuk tim pengembang sebelum pengembangan skala besar dimulai.



Gambar 2.2 Pendekatan *Prototipe*
(Sumber: Rosa, dkk, 2011:30)

Rosa,dkk (31:2011) juga mengungkapkan : “Model prototype cocok digunakan untuk menjabarkan kebutuhan pelanggan secara lebih detail karena pelanggan sering kali kesulitan dalam menyampaikan kebutuhannya secara detail tanpa melihat gambaran yang jelas”.

Untuk mengantisipasi agar proyek dapat berjalan sesuai dengan target waktu dan biaya dari awal , maka sebaiknya spesifikasi kebutuhan sistem harus sudah disepakati oleh pengembang dengan pelanggan secara tertulis. Model prototype ini kurang cocok untuk aplikasi dengan slala besar karna membuat prototype untuk aplikasi skala besar akan sangat memakan waktu dan tenaga.

2.2 Komponen Sistem Informasi

Sistem informasi terdapat komponen-komponen seperti (Kadir, 2007):

1. Perangkat keras *(hardware)*
   Mencakup peranti-peranti fisik seperti komputer dan printer.
2. Perangkat lunak *(software)*
   Sekumpulan instruksi yang memungkinkan perangkat keras untuk dapat memproses data.
3. Prosedur
   Sekumpulan aturan yang dipakai untuk mewujudkan pemrosesan data dan pembangkitan keluaran yang dikehendaki.
4. Orang
   Semua pihak yang bertanggung jawab dalam pengembangan sistem informasi, pemrosesan, dan penggunaan keluaran sistem informasi.
5. Basis data *(database)*
   Sekumpulan tabel, hubungan dan lain-lain yang berkaitan dengan penyimpanan data.
6. Jaringan komputer dan komunikasi data
   Sistem penghubung yang memungkinkan sumber *(resources)* dipakai secara bersama atau diakses oleh sejumlah pemakai.

2.3 Konsep Analisa dan Perancangan Sistem Informasi

Tahapan analisis sistem merupakan tahapan yang sangat penting karena kesalahan didalam tahapan ini akan menyebabkan kesalahan pada tahapan selanjutnya. Proses analisa sistem dalam pengembangan sistem informasi merupakan suatu prosedur yang dilakukan untuk pemeriksaan masalah dan

penyusunan pemecahan masalah yang timbul serta membuat spesifikasi sistem yang baru (Jogiyanto, 2009).

Analisis sistem mencakup (Jogiyanto, 2009):

1. Analisa kelayakan, merupakan proses yang mempelajari atau menganalisa permasalahan yang telah ditentukan sesuai dengan tujuan akhir yang akan dicapai.
2. Analisa kebutuhan, merupakan proses untuk menghasilkan spesifikasi kebutuhan.

Secara garis besar tahapan analisa sistem dibagi menjadi empat langkah sebagai berikut :

1. *Identify* (mengidentifikasi masalah)
2. *Understand* (memahami kinerja sistem)
3. *Analyze* (menganalisa sistem)
4. *Report* (membuat laporan)

Tahapan berikutnya adalah tahap perancangan sistem, tahap perancangan sistem adalah sebagai berikut :

1. Perancangan model, rancangan dalam bentuk fisik dan model logika.
2. Perancangan bentuk keluaran, rancangan bentuk-bentuk laporan sistem dan dokumennya.
3. Perancangan bentuk masukkan, rancangan bentuk-bentuk masukkan di dokumen dan di layar ke sistem informasi.
4. Perancangan basisdata (*Database*), rancangan file-file yang dibutuhkan guna penyimpanan data dalam sistem informasi.

2.4 Analisa dan Perancangan Berorientasi Objek

Metode berorientasi objek merupakan paradigm baru dalam rekayasa perangkat lunak yang memandang sistem sebagai sekumpulan objek-objek diskrit yang saling berinteraksi. Yang dimaksud dengan berorientasi objek adalah bahwa mengorganisasikan perangkat lunak sebagai kumpulan objek-objek diskrit yang bekerja sama antara informasi atau struktur data dan perilaku yang mengaturnya (Sholiq, 2006).

Perusahaan *software*, rational *software* telah membentuk konsorium dengan berbagai organisasi untuk meresmikan pemakaiam *Unified Modelilling language* (UML), sebagai bahasa standar dalam *object oriented analysis design (*OOAD).

2.5 Analisa Sistem

Analisa sistem adalah kegiatan untuk melihat sistem yang sudah berjalan, melihat bagian mana yang bagus dan tidak bagus, dan kemudian mendokumentasikan kebutuhan yang akan dipenuhi dalam sistem yang baru. Hal tersebut terlihat sederhana, namun sebenarnya tidak. Banyak hambatan yang akan ditemui dalam proses tersebut (Shalahuddin, 2011).

2.6 *Object Oriented Analysis* (OOA)

Object Oriented Analysis (OOA) adalah tahapan untuk menganalisis spesifikasi atau kebutuhan akan sistem yang akan dibangun dengan konsep beriontasi objek (Rosa dan Salahuddin, 2011).

Adapun aktivitas OOAD adalah (Rosa dan Salahuddin, 2011).

1. Menganalisis masalah domain.
2. Menjelaskan sistem proses.
3. Mengidentifikasi objek.
4. Menentukan atribut.
5. Mengidentifikasi operasi.
6. Komunikasi objek.

2.7 *Object Oriented Design* (OOD)

Object Oriented Design (OOD) adalah perantara untuk memetakan spesifikasi atau kebutuhan sistem yang akan dibangun dengan konsep berorientasi objek ke desain pemodelan agar lebih mudah diimplementasikan dengan pemrograman berorientasi objek. OOA dan OOD seringkali memiliki batasan yang sama, sehingga biasanya disebutkan langsung menjadi OOAD (Rosa dan Salahuddin, 2011).

1. Mendefenisikan konteks dan mode dari penggunaan sistem.
2. Mendesain struktur sistem.
3. Identifikasi objek sistem utama.
4. Mengembangkan model desain.
5. Menetukan interface objek

2.8 *Object-oriented Analysis and Design* (OOAD)

OOAD adalah metode pengembangan sistem yang lebih menekankan objek dibandingkan dengan data atau proses. Ada tiga ciri khas dari pendekatan ini, yaitu:

a. Object adalah struktur yang mengenkapsulasi atribut dan metode yang beroperasi berdasarkan atribut-atribut tadi. Objek adalah abstaksi dari benda nyata dimana data dan proses diletakkan bersama untuk memodelkan struktur dan perilaku dari objek dunia nyata.
b. Object class adalah sekumpula objek yang berbagi stuktur yang sama dan perilaku yang sama.
c. *Inheritance,* merupakan properti yang muncul ketika tipe entitas atau object class disusun secara hierarki dan setiap tipe entitas atau menerima atau mewarisi atribut dan metode dari pendahuluannya (Fatta, 2007).

2.9 *Blackbox Testing*

Blackbox Testing (pengujian kotak hitam) Merupakan pengujian perangkat lunak dari segi spesifikasi fungsional tanpa menguji desain dan kode program. Pengujian dimaksudkan untuk mengetahui apakah fungsi-fungsi, masukan, dan keluaran dari perangkat lunak sesuai dengan spesifikasi yang dibutuhkan. (Rosa dan Shalahuddin, 2013)

Pengujian Kotak hitam dilakukan dengan membuat kasus uji yang bersifat mencoba semua fungsi dengan memakai perangkat lunak apakah sesuai dengan spesifikasi yang dibutuhkan. Kasus uji yang dibuat untuk melakukan pengujian kotak hitam harus dibuat dengan kasus benar dan kasus salah, misalkan untuk kasus proses *login* maka kasus uji yang dibuat adalah:

1) Jika *user* memasukkan nama pemakai (*username)* dan kata sandi yang benar (*password)* yang benar.
2) Jika *user* memasukkan *username* dan *password* yang salah, misalnya *username* benar tapi *password* salah, atau sebaliknya, atau keduanya salah.

2.9.1 Cakupan Pengujian

Cakupan pengujian yang dilakukan pada *Blackbox testing* adalah perihal pengujian *interface* dan *form validation*. Pengujian *interface* adalah pengujian yang dilakukan secara langsung terhadap desain *interface* yang dibuat pada sistem.

2.9.2 Tujuan Pengujian

Tujuan yang diharapkan dalam melakukan *Blackbox Testing* adalah dapat membuat desain dan fungsi sistem yang sesuai dengan kebutuhan organisasi, lembaga atau perusahaan.

2.10 Sekilas tentang *Cloud Storage*

2.10.1 Definisi *Cloud Storage*

Komputasi Awan pada bagian Penyimpanan Awan (*Cloud Storage*) berasal dari Teori awal "*Cloud Attached Storage*" yang merupakan pengembangan dari layanan yang berada di ruang lingkup yang sederhana, yaitu dari ruang lingkup jaringan lokal atau biasa dikenal dengan sebutan *Network Attached Storage (NAS).* Layanan *Network Attached Storage* memiliki ruang lingkup yang sederhana pada jaringan yang kecil (Pradip, 2013).

Jadi dengan kata lain, *Cloud Storage* adalah sebuah teknologi penyimpanan data digital yang memanfaatkan adanya *server virtual* sebagai media penyimpanan. Tidak seperti media penyimpanan perangkat keras pada umumnya seperti *compact disk* atau *hardisk*, teknologi *Cloud storage* tidak membutuhkan perangkat tambahan apapun. Yang anda perlukan untuk mengakses file digital anda hanyalah perangkat komputer atau gadget yang telah dilengkapi layanan internet. (Ahmad, 2016).

Memang pada dasarnya terdapat banyak pilihan untuk menyimpan data-data. Dapat disimpan pada media penyimpanan fisik seperti *harddisk*, *CD*, *flashdisk*. Tapi untuk sebagian pengguna komputer hal tersebut mulai menjadi penghalang efisiensi kerja saat data-data yang dibutuhkan tak dapat diakses. Misalnya *flashdisk* yang tertinggal atau keping *CD* yang rusak. Saat data disimpan secara '*cloud*', data-data tersebut dapat dengan mudah diakses lewat jaringan internet. Tidak perlu mengambil laptop atau *flashdisk*. Yang perlu ditekankan dari keuntungan teknologi ini adalah kemudahan mengakses data kita dimana saja, kapan saja, dan menggunakan perangkat apa saja (Firdauska, 2014).

2.10.2 Kelebihan *Cloud Storage*

Berikut beberapa kelebihan dari *Cloud Storage* , diantaranya adalah: (Pradip, 2013)

a. *Skalabilitas*, yaitu dengan *cloud storage* bisa menambah kapasitas penyimpanan data tanpa harus membeli peralatan tambahan, seperti *hardisk* dll. Cukup menambah kapasitas yang disediakan oleh penyedia layanan *cloud computing*.

b. Aksesibilitas, yaitu bisa dengan mudah mengakses data kapanpun dan dimanapun *user* berada, asal user tersebut terkoneksi dengan internet, sehingga memudahkan *user* untuk mengakses data disaat yang penting.

c. Keamanan, yaitu data bisa terjamin keamanan nya oleh penyedia layanan *cloud computing*, sehingga bagi perusahaan yang berbasis IT, data bisa disimpan secara aman di penyedia *cloud computing*. Itu juga mengurangi biaya yang diperlukan untuk mengamankan data perusahaan.

d. Kreasi, yaitu para user bisa melakukan/mengembangkan kreasi atau project mereka tanpa harus mengirimkan project mereka secara langsung ke perusahaan, tapi user bisa mengirimkan nya lewat penyedia layanan *cloud computing*.

e. Kecemasan, ketika terjadi bencana alam data milik kita tersimpan aman di *cloud* meskipun *hardisk* atau gadget dalam keadaan rusak.

2.10.3 Kekurangan *Cloud Storage*

Sedangkan kekurangan dari teknologi *cloud storage* ini diantaranya : (Pradip, 2013)

a. Terletak pada hal keamanan, setiap akun dilindungi oleh password yang bisa saja diketahui orang lain jika tidak memperhatikan keamanan pengaksesannya, walaupun sudah begitu berhati-hati tetap tidak menutup kemungkinan jika akun dapat dibuka orang yang tidak bertanggung-jawab.
b. Gangguan pada saat mengakses data, bisa disebabkan karena koneksi yang bermasalah atau server yang sedang *down*, beberapa orang menggunakan teknologi ini untuk menyimpan data-data yang sering digunakan dalam pekerjaan, saat mereka berpindah tempat mereka tak perlu bersusah payah mem-*backup* atau membawa data-data yang diperlukan untuk bekerja di tempat yang baru, sebagian orang juga menggunakan teknologi ini untuk menyimpan data penting sebagai cadangan (*backup*) yang sewaktu-waktu tanpa diketahui bisa rusak ataupun hilang.

2.11 Sekilas Tentang UML

2.11.1 *Unified Modelling Language (UML)*

Menurut Sugiarti (34:2005) "*Unified Modelling Language (UML)* adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi dalam merancang dan mendokumentasikan sistem piranti lunak". UML menawarkan sebuah standart untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax / semantik.* Notasi

UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan.

2.11.2 Jenis – Jenis Diagram UML

2.11.2.1 *Use Case Diagram*

Menurut Sugiarti (41:2005) *Use Case diagram* adalah merupakan pemodelan untuk menggambarkan kelakuan (behavior) system yang akan dibuat. Diagram use case mendeskripsikan sebuah interaksi antara satu atau lebih actor dengan system yang akan dibuat. Dengan pengertian yang cepat, diagram use case digunakan untuk mengetahui fungsi apa saja yang ada didalam sebuah system dan siapa saja yang berhak menalankan funsi-fungsi tersebut.

Use Case diagram dapat sangat membantu apabila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

Tabel 2.1 Use case Diagram

NO	GAMBAR	NAMA	KETERANGAN
1		*Use Case*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu actor
2		*Actor*	Menspesifikasikan himpuan peran yang pengguna mainkan ketika berinteraksi dengan *use case*.
3		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.
4		<<*extend*>>	Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan.

5		*Generalization*	Hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*).
6		<<*include*>>	Menspesifikasikan bahwa *use case* sumber secara *eksplisit*.

(Sumber: Rosa dkk, 131:2011)

2.11.2.2 *Class Diagram*

Menurut Sugiarti (57:2012) "*Class diagram* mendeskripsikan jenis-jenis objek dalam sistem dan berbagai hubungan statis yang terdapat di antara mereka". *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class Diagram* menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok:

1. Nama (dan stereotype)
2. Atribut
3. Metoda

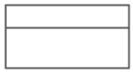
Atribut dan metoda dapat memiliki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
3. *Public*, dapat dipanggil oleh siapa saja

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*. *Class*

dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*.

Tabel 2.2 Class Diagram

NO	GAMBAR	NAMA	KETERANGAN
1		*Class*	Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
2		*Interface*	Sama dengan konsep *interface* dalam pemrograman berorientasi objek.
3		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.
4		*Directed association*	Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity*.
5		*Generalization*	Hubungan dimana objek anak *(descendent)* berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*).
6		*Dependency*	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempegaruhi elemen yang bergantung padanya elemen yang tidak mandiri

(Sumber: Rosa, dkk, 123:2011)

2.11.2.3 *Activity Diagram*

Menurut Sugiarti (2012) "*Activity Diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir

berawal, *decisio*n yang mungkin terjadi, dan bagaimana mereka berakhir". *Activity diagram* juga dapat menggambarkan proses parallel yang mungkin terjadi pada beberapa eksekusi.

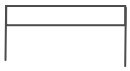
Activity diagram menggambarkan *workflow* (aliran kerja) atau aktifitas dari sebuah sistem atau proses bisnis (Rosa, dkk, 2011). Yang perlu diperhatikan disini adalah bahwa *diagram aktivity* menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir.

1. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.
2. *Activity diagram* merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*).
3. *Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.

Tabel 2.3 Simbol-Simbol Activity Diagram

NO	GAMBAR	NAMA	KETERANGAN
1		Status Awal	Bagaimana objek dibentuk atau diawali.
2		*Aktivitas*	Aktifitas yang dilakukan sistem, aktifitas biasanya diawali dengan kata kerja.
3		*Decision*	Asosiasi percabangaan dimana jika ada pilihan aktifitas lebih dari satu.

4		*Join*	Asosiasi penggabungan dimana lebih dari satu aktivitas digabung menjadi satu.
5		*Status Akhir*	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki akhir.
6		*Swimlane*	Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang ada

(Sumber: Rosa,dkk, 134:2011)

2.11.2.4 *Sequence Diagram*

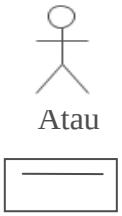
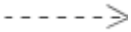
Menurut Sugiarti (69:2012) "*Sequence Diagram* menggambarkan kelakuan/perilaku objek padause case dengan mendeskripsikan waktu hidup objek dan message yang yang dikirimkan dan diterima antar objek". Oleh karna itu untuk menggambarkan diagram sequence maka harus diketahui objek-objek yang terlibat dalam sebuah use case beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram squence yang harus digambar adalah sebanyak pendefinisian use case yang memiliki proses sendiri atau yang penting semua use case yang telah didefinisikan interaksi jalanya pesan sudah dicakup pada diagram sequence sehingga semakin banyak use case yang didefinisikan maka diagram sequence yang harus dibuat juga semakin banyak.

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu.

1. *Sequence diagram* terdiri antar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).
2. *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu, dapat dilihat pada table 2.4 berikut ini :

Tabel 2.4 Simbol-Simbol Sequence Diagram

NO	GAMBAR	NAMA	KETERANGAN
1	 Atau	*Aktor*	Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem.
2	.....	*Lifeline*	Menyatakan kehidupan suatu objek
3		Objek	Merupakan objek yang berinteraksi pesan.
4		Waktu aktif	Menyatakan objek dalam keadaan aktif dan beriteraksi pesan.
5		Pesan tipe *create*	Menyatakan suatu objek memuat objek yang lain, arah panah mengarah pada objek yang dibuat.
6		Pesan tipe *call*	Menyatakan suatu objek memanggil oprasi/metode yang ada pada objek lain atau sendirinya sendiri.
7		Pesan tipe *send*	Menyatakan bahwa suatu objek mengirimkan data /masukan/informasi ke objek lainnya.
8		Pesan tipe *return*	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatu kembali ke objek tertentu.

9		Pesan tipe *destroy*	Menyatakan suatu objek mengakhiri hidup objek yang lain.

(Sumber: Rosa, dkk, 138:2011)

2.11.3 Langkah-Langkah Penggunaan UML secara umum :

Menurut Sugiarti (81:2012) berikut adalah langkah-langkah penggunaan UML secara umum :

a. Langkah pertama adalah membuat daftar *business process* dari level tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
b. Selanjutnya *Use Case* untuk tiap *business process* dipetakan untuk mendefinisikan dengan tepat fungsionalitas yang harus disediakan oleh system. Dan *Use Case diagram* diperhalus dan dilengkapi dengan *requirement, constraints* dan catatan-catatan lain.
c. Fungsi *Deployment Diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
d. Pendefinisian *requirement* lain (*non-fungsional, security* dan sebagainya) yang juga harus disediakan oleh sistem.
e. Berdasarkan *Use Case diagram*, mulailah membuat *activity diagram*.
f. Diperlukan adanya definisi objek-objek level atas (*package* atau *domain*) kemudian pembuatan *sequence* atau *Collaboration Diagram* untuk tiap alir pekerjaan. Jika sebuah *Use Case* memiliki kemungkinan alir normal dan *error*, perlu dibuat satu diagram untuk masing-masing alir.
g. Selanjutnya diperlukan adanya rancangan *user Interface* model yang menyediakan antarmuka bagi pengguna untuk menjalankan skenario *Use Case*.

h. Berdasarkan model-model yang sudah ada, dapat dibuat *Class Diagram*. Setiap *package* atau *domain* dipecah menjadi *hirarki class* lengkap dengan atribut dan metodanya. Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji *fungsionalitas class* dan interaksi dengan *class* lain.

i. Setelah *Class Diagram* dibuat, kita dapat melihat kemungkinan pengelompokkan *class* menjadi komponen komponen. Karena itu perlu dibuatnya *Component Diagram* pada tahap ini. Juga, diperlukan adanya definisi tes integrasi untuk setiap komponen meyakinkan ia berinteraksi dengan baik.

j. Perhalus *Deployment Diagram* yang sudah dibuat. Detailkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan, dan sebagainya. Petakan komponen ke dalam *node*.

k. Setelah tahap-tahap diatas baru dapat dimulai membangun sistem. Ada dua pendekatan yang dapat digunakan :

   1. Pendekatan *Use Case*, dengan meng-*assign* setiap *Use Case* kepada tim pengembang tertentu untuk mengembangkan unit *code* yang lengkap dengan tes.
   2. Pendekatan komponen, yaitu meng-*assign* setiap komponen kepada tim pengembang tertentu.

Apabila tahap-tahap diatas telah terpenuhi maka diperlukan adanya uji modul dan uji integrasi serta perbaikan model beserta *code*-nya. Model harus selalu sesuai dengan *code* yang aktual.

2.12 *Hypertext Preprocessor (PHP)*

2.12.1 Definisi *PHP*

Singkatan dari *PHP* yaitu *Hypertext Preprocessor* yang digunakan sebagai bahasa script *server*-side dalam pengembangan *web* yang disisipkan pada dokumen *HTML*. Penggunaan *PHP* memungkinkan *Web* dapat dibuat dinamis sehingga *maintenance* situs *web* tersebut menjadi lebih mudah dan efisien. *PHP*

merupakan *software open-source* yang disebarkan dan dilisensikan secara gratis serta dapat didownload secara bebas dari situs resminya *http://www.php.net*, *PHP* ditulis dengan menggunakan bahasa C (Peranginangin, 2006).

2.12.2 Kelebihan *PHP*

PHP dapat digunakan pada semua sistem operasi , antara lain *Linux, Unix* (termasuk variannya *HP-UX*, *Solaris*, dan *Open BSD*), *Microsoft Windows*, *MacOS X*, *RISC OS*. *PHP* juga mendukung banyak *web server*, seperti *Apache*, *Microsoft Internet Information Server (MIIS)*, *Personal Web Server (PWS)*, *Netscape and iPlanet Servers*, *Oreilly website Pro Server*, *audium*, *Xitami*, *OmniHTTPd*, dan masih banyak lagi lainnya, bahkan *PHP* dapat bekerja sebagai *CGI Processor* (Peranginangin, 2006).

PHP tidak terbatas pada hasil dari keluaran *HTML* (*Hypertext Markup Languages*). *PHP* juga memiliki kemampuan untuk mengolah keluaran gambar, file *PDF*, dan *movies Flash*. *PHP* juga dapat menghasilkan teks seperti *XHTML* dan file *XML* lainnya.

2.13 Database Server MySQL

2.12.1 Pengertian MySql

Menurut Nugroho (243:2004) “untuk dapat menghubungkan database dengan bahasa pemograman, *MySql* memiliki sebuah dukungan fungsi API yang berguna untuk melakukan hubungan antara database dengan program”. Banyak sekali dukugan yang dimiliki database *MySql* dalam hubungan program, hamper semua bentuk bahasa pemograman dapat memanfaatkan database ini sebagai media penyimpanan datanya.

MySQL sebenarnya merupakan turunan salah satu konsep utama dalam database sejak lama yaitu, SQL (*Structured Query Language*). SQL adalah sebuah konsep pengoperasian database, terutam untuk pemilihan/seleksi dan pemasukan data, yang memungkinkan pengoperasian data dikerjakan dengan mudah secara otomatis. Keandalan sustu sistem database (DBMS) dapat diketahui dari cara kerja *optimizer*-nya dalam melakukan prose perintah perintah SQL, yang dibuat

oleh user maupun program program aplikasinya. Sebagai databse server, *MySQL* dapat dikatakan lebih unggul dibandingkan database server lainnya dalam query data. Hal ini terbukti untuk query yang dilakukan oleh single user, kecepatan *query MySQL* bisa sepuluh kali lebih cepat dari *PostgreSQL* dan lima kali lebih cepat dibandingkan Interbase. Kemmpuan yang cukup menakjubkan untuk sebuah software gratisan.

MySQL adalah satu dari sekian banyak sitem database, merupakan terobosan solusi yang tepat dalam aplikasi database. Didukung oleh ribuan bahkan jutaan komunitas pengguna di internet yang siap membantu. Selain itu juga tersedia mailing list dan homepage khusus yang memberikan tutorial serta dokumen lengkap.

2.12.2 Sejarah Singkat Tentang MySQL

MySQL dikembangkan sekitar tahun 1994 oleh sebuah perusahaan pengembang software dan konsultan database yang bernama MySQL AB yang bertempat di SWEDIA.Waktu itu perusahaan tersebut masih bernama TcX DataKonsult AB, dan tujuan awal dikembangkannya MySQL adalah untuk mengembangkan aplikasi berbasis web pada client. Awalnya Michael Widenius “Monty”, pengembang satu-satunya di TcX, memiliki sebuah aplikasi *UNIREG* dan rutin *ISAM* buatannya sendiri sedang mencari antar muka SQL yang cocok unutk diimplementasikan ke dalamnya. Mula-mula Monty memakai miniSQL(mSQL) pada eksperiment itu, namun mSQL dirasa kurang sesuai, karena terlalu lambat dalam pemrosesan *query*.

Akhirnya Monty menghubungi David Hughes, pembuat mSQL yang sedang merilis versi kedua dari mSQL. Kemudian Monty mencoba membuat sendiri mesin SQL yang memilki antarmuka mirip dengan SQL, tetapi dengan kemampuan yang lebih sesuai, dan lahirlah MySQL.

2.12.3 Keistimewaan MySQL

Berikut ini beberapa keistimewaan yamg dimilki oleh MySQL:

1. *Portability*

   MySQL dapat berjalan stabil pada berbagi sistem operasi diantaranya adalah seprti Windows, Linux, FreeBSD, Mac OS X Server, Amiga, HP-UX dan masih banyak lagi.

2. *Open Source*

   MySQL didstribusikan secara *open source* (gratis), di bawah licensi GPL sehingga kita dapat menggunkannya secara cuma-cuma tanpa dipungut biaya sepeser pun.

3. *Multiuser*

   MySQL dapat digunakan oleh beberapa user dalam waktu yang bersamaan tanpa mengalami masalah atau konflik. Hal ini memungkin sebuah database server MySQL dapat diakses client secara bersamaan.

4. *Performance Tuning*

   MySQL memilki kecepatan yang menkajubkan dalam menangani query sederhana, dengankata lain dapat memproses lebih banyak SQL per satuan waktu.

5. *Column Types*

   MySQL memilki tipe kolom yang sangat kompleks, seperti integer, float, double, char, varchar, text, blob, date, time, datetime, timestamp, year, set serta enum.

6. *Command dan Functions*

   MySQL memilki operator dan fungsi yang secara penuh yang mendukung perintah SELECT dan WHERE dalam query.

7. *Security*

   MySQL memiliki beberapa lapisan sekuritas seperti level subnetmask, nama host, dan izin akses user dengan sistem perizinan yang mendetail serta password terenkripsi.

8. *Scalability dan Limits*

   MySQL mampu menangani databse dalam skala besar, dengan jumlah records lebih dari 50 juta dan 60 ribu tabel serta 5 miliar baris. Selain itu, batas indeks yang dapat ditampung mencapai 32 indeks pada tipa tabelnya .

9. *Conectivity*

   MySQL dapat melakukan koneksi dengan client dengan menggunakan *protocol TCP/IP, Unix soket(unix),* atau *Named Pipes(NT).*

10. *Localisation*

    MySQL dapat mendeteksi pesan kesalahan *(error code)* pada client dengan menggunakan lebih dari dua puluh bahasa.

11. *Interface*

    MySQL memiliki *interface* terhadap berbagai aplikasi dan bahasa pemrograman.

12. *Client dan Tools*

    MySQL dilengkapi dengan berbagai tool yang dapat digunakan untuk administrasi database, dan setiap tool yang ada disediakan petunjuk online.

13. *Struktur Table*

    MySQL memiliki struktur table yang lebih fleksibel dalam menangani ALTER TABLE, dibandingkan database lainnya semacam *ProstgreSQL* ataupun *Oracle.*

2.13 Gambaran Umum Perusahaan

Yayasan Perintis Sumbar yang sudah berdiri sejak 21 Juli 1988 dengan AKTA No. 88, dan sudah terdaftar pada Kementerian Hukum dan Hak Asasi Manusia Republik Indonesia Nomor: AHU – 1248.AH.01.04.Tahun 2011. Tujuannya adalah membantu pemerintah dalam melaksanakan upaya-upaya pendidikan tenaga kesehatan.

2.14 Sejarah Perusahaan

Yayasan Perintis Sumbar yang sudah berdiri sejak 21 Juli 1988 dengan AKTA No. 88, dan sudah terdaftar pada Kementerian Hukum dan Hak Asasi Manusia Republik Indonesia Nomor: AHU – 1248.AH.01.04.Tahun 2011. Tujuannya adalah membantu pemerintah dalam melaksanakan upaya-upaya pendidikan tenaga kesehatan.

Pada awal berdiri sekolah yang dikelola Yayasan Perintis adalah Sekolah Menengah Analis Kesehatan (SMAK) di Padang tahun 1988 dan Akademi Keperawatan (AKPER) Perintis yang berdiri pada tahun 1989 di Bukittinggi. Enam tahun kemudian SMAK dikonversi ke Akademi Analis Kesehatan (AAK) setara D-III sesuai Surat Keputusan Menteri Kesehatan RI No. HK 00.06.13.5946 tahun 1995 dan melalui Izin Mendikbud RI No. SK. 082/D/O/1995 berdiri pula Akademik Gizi (AKZI). Pada tahun 1997 Yayasan Perintis mendirikan Sekolah Tinggi Farmasi Indonesia (STIFI) di Padang.

Pada tahun 2006 berdiri Sekolah Tinggi Ilmu Kesehatan (STIKes) Perintis Sumatera Barat (Sumbar) melalui izin Kepmendiknas No. 162/D/O/2006 dengan dua progam studi yaitu S-I Keperawatan dan D-III Kebidanan di Bukittinggi. Pada Tahun 2007 melalui Kepmendiknas No. 17/D/O/2007 untuk program studi S-I Gizi dan D-IV Analis Kesehatan di Padang danpengabungan Akademi: Keperawatan, Gizi dan Analis dibawah Sekolah Tinggi Ilmu Kesehatan (STIKes) Perintis Sumatera Barat. Sekarang delapan program studi yang ada dibawah naungan STIKES Perintis Suamatera Barat, dan semua Program Studi telah terakreditasi.

STIKES Perintis Sumbar memiliki dua kampus dengan kantor pusat administrasi dan sekaligus sebagai Kampus I yang berlokasi di Jl. Adinegoro Km. 17 Simpang Kalumpang Lubuk Buaya Padang Sumatera Barat.

2.15 Visi dan Misi STIKes Perintis

Visi dan misi STIKes Perintis Bukit Tinggi yaitu:

2.15.1 Visi

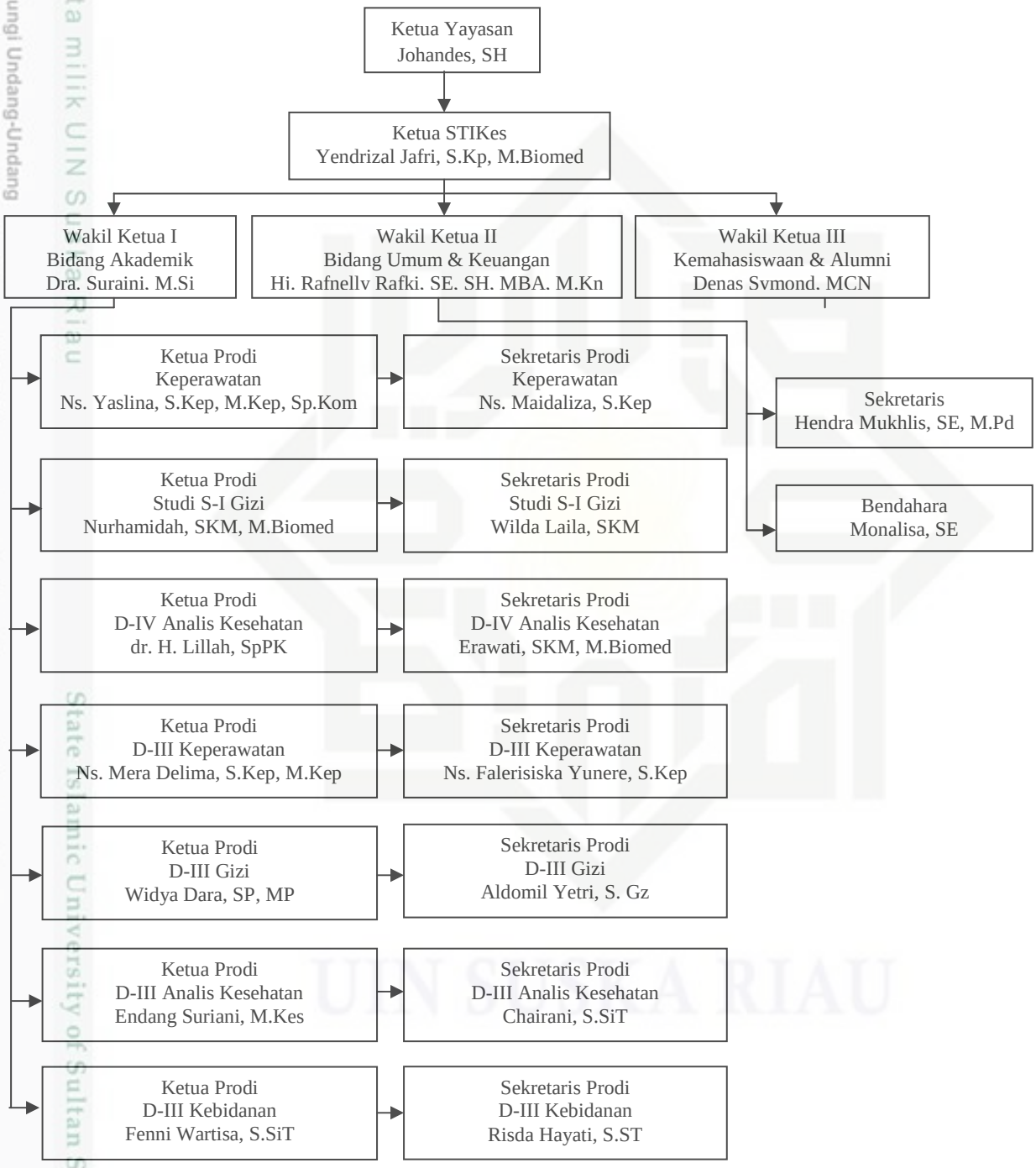
Visi STIKES Perintis Sumbar adalah menjadi Sekolah Tinggi Ilmu Kesehatan yang sehat dan bermutu pada tahun 2020.

2.15.2 Misi

1. Menyelenggarakan pendidikan kesehatan yang berkualitas dan berkesinambungan
2. Menyelenggarakan pendidikan kesehatan yang berbasis ilmu pengetahuan & teknologi dan berakhlak mulia.
3. Mengembangkan kegiatan pendidikan, penelitian dan pengabdian masyarakat yang bermutu.
4. Mengembangkan organisasi dalam meningkatkan kualitas tatakelola yang baik (good university governance).

2.17 Struktur Organisasi Perusahaan

Bagan struktur organisasi pada STIKes Perintis Bukit Tinggi adalah sebagai berikut :



Gambar 2.3 Struktur Organisasi STIKes Perintis Bukit Tinggi